capítulo

4

La estrategia divide-y-vencerás

Esta estrategia constituye un poderoso paradigma para definir algoritmos eficientes. Este método primero divide un problema en dos subproblemas más pequeños de modo que cada subproblema sea idéntico al problema original, excepto porque su tamaño de entrada es menor. Luego, ambos subproblemas se resuelven y las subsoluciones se fusionan en la solución final.

Una cuestión muy importante sobre esta estrategia es que divide el problema original en dos subproblemas idénticos al problema original. Por lo tanto, también estos dos subproblemas pueden resolverse aplicando la estrategia divide-y-vencerás. O, para decirlo con otras palabras, se resuelven de manera recurrente o recursiva.

Consideremos el sencillo problema de encontrar el máximo (número mayor) de un conjunto S de n números. La estrategia divide-y-vencerás resolvería este problema al dividir la entrada en dos conjuntos, cada uno con $n/2$ números. Sean S_1 y S_2 estos dos conjuntos. Luego se encuentra el máximo de S_1 y S_2, respectivamente. Sea X_i, $i = 1, 2$, el máximo de S_i. Así, el máximo de S puede encontrarse al comparar X_1 y X_2. El que sea mayor de éstos será el máximo de S.

En el análisis anterior casualmente se mencionó que es necesario encontrar el máximo X_i. Pero, ¿cómo se encuentra X_i? Nuevamente puede aplicarse la estrategia divide-y-vencerás. Es decir, S_i se divide en dos subconjuntos, se encuentran los máximos de estos subconjuntos y después de fusionan los resultados.

En general, un algoritmo divide-y-vencerás consta de tres pasos.

Paso 1. Si el tamaño del problema es pequeño, éste se resuelve aplicando algún método directo; en caso contrario, el problema original se separa en dos subproblemas, de preferencia del mismo tamaño.

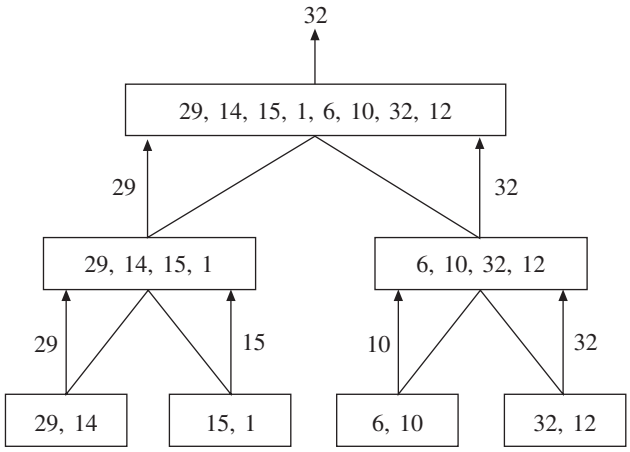
Paso 2. Los dos subproblemas se resuelven de manera recurrente aplicando el algoritmo divide-y-vencerás a los subproblemas.

Paso 3. Las soluciones de los subproblemas se fusionan en una solución del problema original.

La recurrencia del paso 2 crea subproblemas cada vez más pequeños. Al final estos subproblemas son tan pequeños que cada uno puede resolverse directa y fácilmente.

A continuación se mostrará el significado de la estrategia divide-y-vencerás mediante su aplicación para resolver el problema de encontrar el máximo. Imagine que se tienen ocho números: 29, 14, 15, 1, 6, 10, 32 y 12. La estrategia divide-y-vencerás encuentra el máximo de estos ocho números, lo cual se muestra en la figura 4-1.

FIGURA 4-1 Estrategia divide-y-vencerás para encontrar el máximo de ocho números.



Como se muestra en la figura 4-1, primero se separan los ocho números en dos subconjuntos y luego cada uno de ellos se divide en dos subconjuntos que constan de dos números. Debido a que un subconjunto sólo contiene dos números, este subconjunto ya no se divide más. El máximo se determina con una simple comparación de estos dos números.

Una vez que se determinan los cuatro máximos, comienza la fusión. Así, 29 se compara con 15 y 10 se compara con 32. Finalmente, al comparar 32 y 29 se encuentra que el máximo es 32.

En general, la complejidad $T(n)$ de un algoritmo divide-y-vencerás se determina con la siguiente fórmula:

$$T(n) = \begin{cases} 2T(n/2) + S(n) + M(n) & n \geq c \\ b & n < c \end{cases}$$

donde $S(n)$ denota los pasos temporales necesarios para dividir el problema en dos subproblemas, $M(n)$ denota los pasos temporales necesarios para fusionar las dos subsoluciones y b es una constante. Para el problema de encontrar el máximo, la separación y la fusión requieren un número constante de pasos. En consecuencia, se tiene

$$T(n) = \begin{cases} 2T(n/2) + 1 & n > 2 \\ 1 & n = 2. \end{cases}$$

Si se supone que $n = 2^k$, se tiene

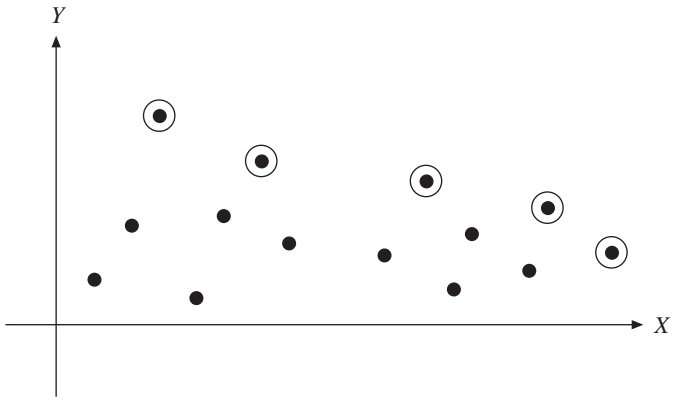
$$\begin{aligned} T(n) &= 2T(n/2) + 1 \\ &= 2(2T(n/4) + 1) + 1 \\ &\vdots \\ &= 2^{k-1}T(2) + \sum_{j=0}^{k-2} 2^j \\ &= 2^{k-1} + 2^{k-1} - 1 \\ &= 2^k - 1 \\ &= n - 1. \end{aligned}$$

El lector debe observar que la estrategia divide-y-vencerás se aplicó dos veces para encontrar el máximo sólo con fines ilustrativos. Resulta evidente que en este caso la estrategia divide-y-vencerás no es demasiado efectiva porque con una exploración directa de estos n números y al hacer $n - 1$ comparaciones también se encuentra el máximo y se tiene la misma eficacia que con la estrategia divide-y-vencerás. No obstante, en los siguientes apartados se demostrará que en muchos casos, sobre todo en problemas geométricos, la estrategia divide-y-vencerás es realmente buena.

4-1 El problema de cómo encontrar puntos máximos en un espacio bidimensional

En el espacio bidimensional se dice que un punto (x_1, y_1) domina a (x_2, y_2) si $x_1 > x_2$ y $y_1 > y_2$. Un punto se denomina máximo si ningún otro punto lo domina. Dado un conjunto de n puntos, el problema de encontrar los máximos consiste en encontrar todos los puntos máximos de entre estos n puntos. Por ejemplo, los puntos encerrados en un círculo en la figura 4-2 son puntos máximos.

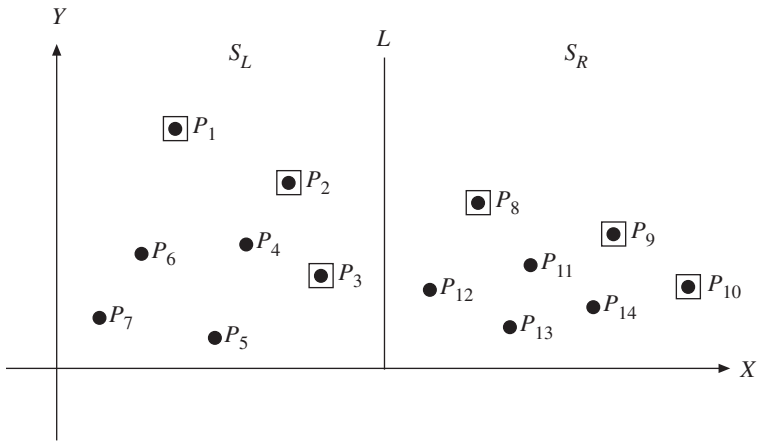
FIGURA 4-2 Puntos máximos.



Un método directo para resolver el problema de encontrar los máximos consiste en comparar cada par de puntos. Este método directo requiere $O(n^2)$ comparaciones de puntos. Como se mostrará a continuación, si se aplica la estrategia divide-y-vencerás, el problema puede resolverse en $O(n \log n)$ pasos, lo cual constituye, de hecho, una mejora bastante aceptable.

Si se aplica la estrategia divide-y-vencerás, primero se encuentra la recta mediana L perpendicular al eje x, que divide en dos subconjuntos todo el conjunto de puntos, como se muestra en la figura 4-3. Los conjuntos de puntos a la izquierda de L y a la derecha de L se denotan por S_L y S_R, respectivamente.

FIGURA 4-3 Los puntos máximos de S_L y S_R.



El siguiente paso es encontrar en forma recurrente los puntos máximos de S_L y S_R. Como se muestra en la figura 4-3, P_1, P_2 y P_3 son puntos máximos de S_L y P_8, P_9 y P_{10} son puntos máximos de S_R.

El proceso de fusión es muy sencillo. Debido a que el valor x de un punto en S_R siempre es mayor que el valor x de todo punto en S_L, un punto en S_L es un máximo si y sólo si su valor y no es menor que el valor y de un máximo de S_R. Por ejemplo, para el caso que se muestra en la figura 4-3, P_3 debe eliminarse porque es dominado por P_8 en S_R. El conjunto de puntos máximos de S es P_1, P_2, P_8, P_9 y P_{10}.

Con base en el análisis anterior, podemos resumir como sigue el algoritmo divide-y-vencerás para resolver el problema de cómo encontrar los puntos máximos en un espacio bidimensional:

Algoritmo 4-1 □ Un método divide-y-vencerás para encontrar puntos máximos en un plano

Input: Un conjunto de n puntos planos.
Output: Los puntos máximos de S.
Paso 1. Si S sólo contiene un punto, proporcionarlo como el máximo. En caso contrario, encontrar una recta L perpendicular al eje x que separe el conjunto de puntos en dos subconjuntos S_L y S_R, cada uno con $n/2$ puntos.
Paso 2. Los puntos máximos de S_L y S_R se encuentran de manera recursiva.
Paso 3. Los puntos máximos de S_L y S_R se proyectan sobre L y se ordenan según sus valores y. Sobre las proyecciones se realiza una exploración lineal y cada uno de los puntos máximos de S_L se elimina si su valor y es menor que el valor y de algún punto máximo de S_R.

La complejidad temporal de este algoritmo depende de las complejidades temporales de los pasos siguientes:

1. En el paso 1 hay una operación que encuentra la mediana de n números. Después se demostrará que esto puede hacerse en $O(n)$ pasos.
2. En el paso 2 hay dos subproblemas, cada uno con $n/2$ puntos.
3. En el paso 3, la proyección y la exploración lineal pueden hacerse en $O(n \log n)$ pasos después de ordenar los n puntos según sus valores y.

Hacer $T(n)$ la complejidad temporal del algoritmo divide-y-vencerás para encontrar puntos máximos de n puntos en un plano. Entonces

$$T(n) = \begin{cases} 2T(n/2) + O(n) + O(n \log n) & n > 1 \\ 1 & n = 1. \end{cases}$$

Sea $n = 2^k$. Es fácil demostrar que

$$\begin{aligned} T(n) &= O(n \log n) + O(n \log^2 n) \\ &= O(n \log^2 n). \end{aligned}$$

Así, se ha obtenido un algoritmo $O(n \log^2 n)$. Es necesario recordar al lector que si se aplica un método directo para revisar cada par de puntos exhaustivamente, se requieren $O(n^2)$ pasos.

Se observa que la estrategia divide-y-vencerás es dominada por el ordenamiento en los pasos de fusión (o mezcla). De alguna manera, ahora no se está realizando un trabajo muy eficiente porque el ordenamiento debe hacerse de una vez por todas. Es decir, debe hacerse un preordenamiento. Si esto se lleva a cabo, entonces la complejidad de la mezcla es $O(n)$ y el número total de pasos necesarios es

$$O(n \log n) + T(n)$$

donde

$$T(n) = \begin{cases} 2T(n/2) + O(n) + O(n) & n > 1 \\ 1 & n = 1 \end{cases}$$

y fácilmente se encuentra que $T(n)$ es $O(n \log n)$. Así, la complejidad temporal total de usar la estrategia divide-y-vencerás para encontrar puntos máximos con preordenamiento es $O(n \log n)$. En otras palabras, se ha obtenido un algoritmo aún más eficiente.

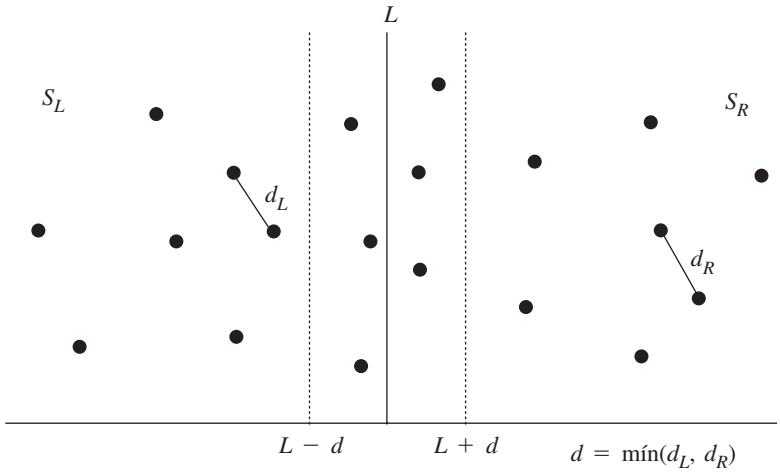
4-2 EL PROBLEMA DEL PAR MÁS CERCANO (CLOSEST NEIGHBOR)

Este problema se define como sigue: dado un conjunto S de n puntos, encontrar un par de puntos que sean los más cercanos entre sí. El problema unidimensional del par más cercano puede resolverse en $O(n \log n)$ pasos al ordenar los n números y realizando una exploración lineal. Al examinar la distancia entre cada par de puntos adyacentes en la lista ordenada es posible determinar el par más cercano.

En el espacio bidimensional la estrategia divide-y-vencerás puede aplicarse para diseñar un algoritmo eficiente a fin de encontrar el par más cercano. Así como se hizo cuando se resolvió el problema de cómo encontrar el máximo, el conjunto S primero se

parte en S_L y S_R de modo que todo punto en S_L esté a la izquierda de todo punto en S_R y el número de puntos en S_L sea igual al número de puntos en S_R. Es decir, se encuentra una recta vertical L perpendicular al eje x de modo que S se corte en dos subconjuntos del mismo tamaño. Al resolver el problema del par más cercano en S_L y S_R, respectivamente, se obtienen d_L y d_R, donde d_L y d_R denotan las distancias de los pares más cercanos en S_L y S_R, respectivamente. Sea $d = \text{mín}(d_L, d_R)$. Si el par más cercano (P_a, P_b) de S consta de un punto en S_L y un punto en S_R, entonces P_a y P_b deben estar en el interior de una franja central en la recta L y acotada por las rectas $L - d$ y $L + d$, como se muestra en la figura 4-4.

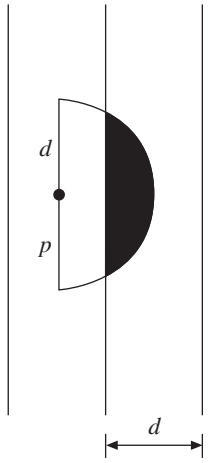
FIGURA 4-4 El problema del par más cercano.



El análisis anterior indica que durante el paso de fusión sólo deben examinarse los puntos que se encuentran en la franja. Aunque en promedio el número de puntos de la franja central no puede ser muy grande, en el peor de los casos habrá no más de n puntos. Así, el método de la fuerza bruta para encontrar el par más cercano en la franja requiere calcular $n^2/4$ distancias y comparaciones. Este tipo de paso de fusión no es bueno para nuestro algoritmo divide-y-vencerás. Afortunadamente, como se demostrará a continuación, el paso de fusión puede completarse en tiempo $O(n)$.

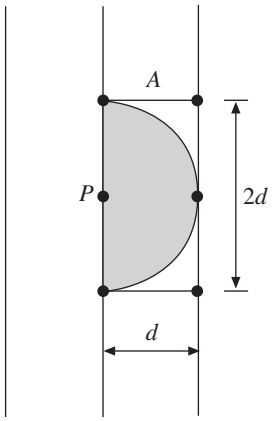
Si un punto P en S_L y un punto Q en S_R constituyen un par más cercano, entonces la distancia entre P y Q debe ser menor que d. Por lo tanto, no es necesario considerar un punto que esté demasiado lejos de P. Considere la figura 4-5. En esta figura basta analizar la región sombreada. Si P está exactamente en la recta L, esta región sombreada es máxima. Incluso en este caso, esta región sombreada estará contenida en el rectángulo A de $d \times 2d$, como se muestra en la figura 4-6. Así, sólo es necesario examinar los puntos dentro de este rectángulo A.

FIGURA 4-5 Región limitada a examinar en el proceso de fusión del algoritmo divide-y-vencerás del par más cercano.



La pregunta que falta es: ¿cuántos puntos hay en el rectángulo A? Como se sabe que la distancia entre cada par de puntos tanto en S_L como en S_R es mayor o igual a d, en este rectángulo sólo puede haber a lo sumo seis puntos, que se muestran en la figura 4-6. Con base en esta observación se sabe que en el paso de fusión, para cada punto en la franja, solamente es necesario examinar un número limitado de puntos en la otra mitad de la franja. Sin pérdida de generalidad, puede suponerse que P está en la mitad izquierda de la franja. Sea y_p el valor y de P. Para P sólo es necesario examinar puntos en la otra mitad de la franja cuyos valores y estén entre $y_p + d$ y $y_p - d$. Como vimos antes, cuando mucho hay seis puntos así.

FIGURA 4-6 Rectángulo A que contiene los posibles vecinos más cercanos de P.



Como ya se analizó, antes de aplicar el algoritmo divide-y-vencerás es necesario ordenar los n puntos según su valor y. Después de hacer el preordenamiento, el paso de fusión se lleva a cabo en sólo $O(n)$ pasos.

A continuación se aplica el algoritmo divide-y-vencerás para resolver el problema bidimensional del par más cercano.

Algoritmo 4-2 □ Un algoritmo divide-y-vencerás para resolver el problema bidimensional del par más cercano

Input: Un conjunto S de n puntos en el plano.

Output: La distancia entre los dos puntos más cercanos.

Paso 1. Ordenar los puntos en S según sus valores y y sus valores x.

Paso 2. Si S contiene un solo punto, regresar ∞ como su distancia.

Paso 3. Buscar una recta mediana L perpendicular al eje x para dividir S en dos subconjuntos del mismo tamaño: S_L y S_R. Todo punto en S_L está a la izquierda de L y todo punto en S_R está a la derecha de L.

Paso 4. Recursivamente aplicar el paso 2 y el paso 3 para resolver los problemas de los pares más cercanos de S_L y S_R. Sea $d_L(d_R)$ la distancia entre el par más cercano en $S_L(S_R)$. Sea $d = \text{mín}(d_L, d_R)$.

Paso 5. Proyectar sobre la recta L todos los puntos dentro de la franja acotada por $L - d$ y $L + d$. Para un punto P en la semifranja acotada por $L - d$ y L, hacer y_p su valor y. Para cada uno de tales puntos P, en la semifranja acotada por L y $L + d$ encontrar todos los puntos cuyo valor y esté entre $y_p + d$ y $y_p - d$. Si la distancia d' entre P y un punto en la otra semifranja es menor que d, sea $d = d'$. El valor final de d es la respuesta.

La complejidad temporal del algoritmo completo es igual a $O(n \log n) + T(n)$ y

$$T(n) = 2T(n/2) + S(n) + M(n)$$

donde $S(n)$ y $M(n)$ son las complejidades temporales del paso de separación en el paso 3 y del paso de fusión en el paso 4. El paso de separación se lleva a cabo en $O(n)$ pasos porque los puntos ya están ordenados según sus valores x. El paso de fusión primero tiene que encontrar todos los puntos dentro de $L - d$ y $L + d$. De nuevo, ya que los puntos están ordenados sobre el eje x, puede llevarse a cabo en $O(n)$ pasos. Para cada punto cuando mucho es necesario examinar otros 12 puntos, 6 de ellos en cada semifranja. Así, esta exploración lineal requiere $O(n)$ pasos. En otras palabras, la complejidad temporal del paso de fusión es $O(n)$.

Así,

$$T(n) = \begin{cases} 2T(n/2) + O(n) + O(n) & n > 1 \\ 1 & n = 1. \end{cases}$$

Puede demostrarse fácilmente que

$T(n) = O(n \log n)$.

La complejidad temporal total es

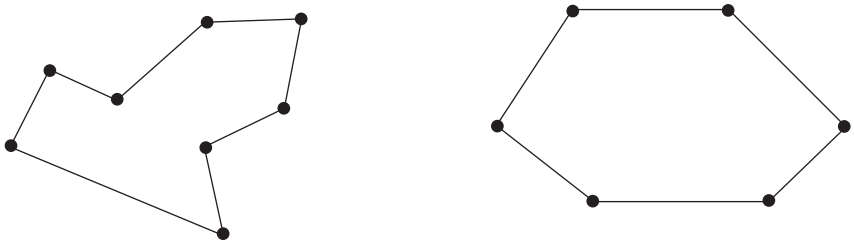
$O(n \log n)$(complejidad del preordenamiento) + $O(n \log n) = O(n \log n)$.

4-3 EL PROBLEMA DEL CONVEX HULL

Este problema se mencionó por primera vez en el capítulo 2. En esta sección se demostrará que este problema del convex hull puede resolverse de manera elegante mediante la estrategia divide-y-vencerás.

Un polígono convexo es un polígono con la propiedad de que cualquier segmento de recta que une dos puntos cualesquiera dentro del polígono debe estar dentro del polígono. En la figura 4-7*a*) se muestra un polígono no convexo y en la figura 4-7*b*) se muestra un polígono convexo.

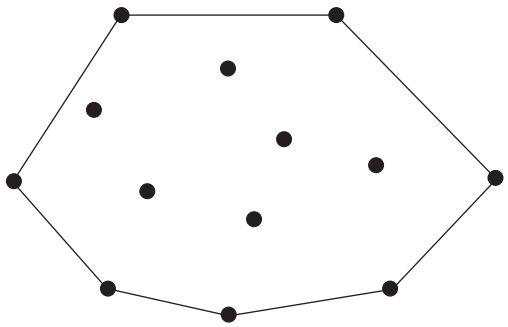
FIGURA 4-7 Polígonos cóncavo y convexo.



a) *b*)

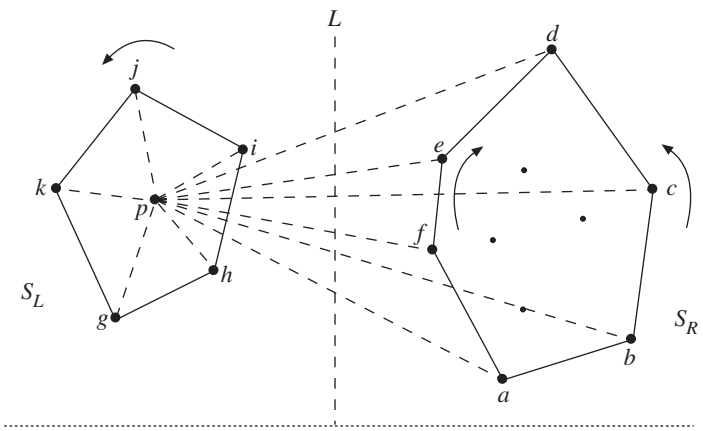
El convex hull de un conjunto de puntos planos se define como el menor polígono convexo que contiene todos los puntos. Por ejemplo, en la figura 4-8 se muestra un convex hull típico.

FIGURA 4-8 Un convex hull.



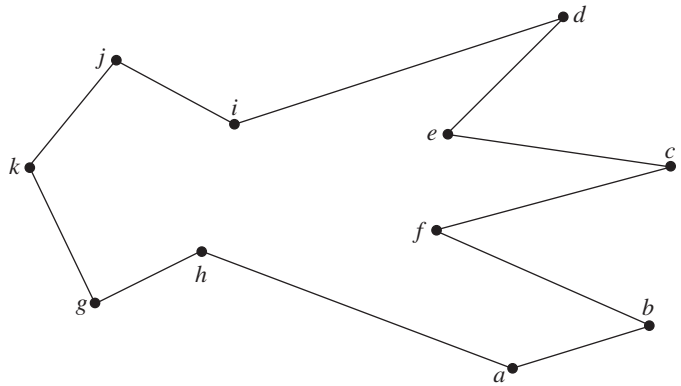
Para encontrar un convex hull puede aplicarse la estrategia divide-y-vencerás. Considere la figura 4-9. En esta figura, el conjunto de puntos planos se ha dividido en dos subconjuntos S_L y S_R por medio de una recta perpendicular al eje x. Luego se construyen conjuntos convexos para S_L y S_R, denominados Hull(H_L) y Hull(H_R), respectivamente. Para combinar Hull(H_L) y Hull(H_R) en un convex hull puede aplicarse el algoritmo de Graham.

FIGURA 4-9 Estrategia divide-y-vencerás para construir un convex hull.



Para llevar a cabo el algoritmo de búsqueda de Graham se escoge un punto interior. Este punto se considera como el origen. Así, cada otro punto forma un ángulo polar con respecto a este punto interior. Luego todos los puntos se ordenan con respecto a estos ángulos polares. El algoritmo de Graham analiza los puntos uno por uno y elimina aquellos que producen ángulos reflexivos, como se ilustra en la figura 4-10. En esta figura se eliminan h, f, e e i. Los puntos restantes son vértices de un convex hull.

FIGURA 4-10 Búsqueda de Graham.



Para llevar a cabo una búsqueda de Graham después que se han construido Hull(S_L) y Hull(S_R), es posible seleccionar un punto interior P de Hull(S_L). Por ser un polígono convexo, debe resultar evidente para el lector que no es necesario examinar los puntos en el interior de los convex hull. Desde la perspectiva de P, Hull(S_R) está en la sección cuyo ángulo origen es menor que o igual a π. Este punto (en forma de cuña) está formado por los dos vértices u y v de Hull(S_R) que pueden encontrarse en tiempo lineal aplicando el siguiente procedimiento: se traza una recta horizontal que pasa por P. Si esta recta corta a Hull(S_R), entonces Hull(S_R) está en la sección determinada por los dos vértices de Hull(S_R) que tienen el mayor ángulo polar $< \pi/2$ y el menor ángulo polar $> 3\pi/2$, respectivamente. Si la recta horizontal que pasa por P no corta a Hull(S_R), la sección está determinada por los vértices que originan el ángulo polar mayor y el menor con respecto a P. En la figura 4-9 estos puntos son a y d. De esta forma, hay tres secuencias de puntos que tienen ángulos polares crecientes con respecto a P, un punto interior de S_L. Las tres secuencias son:

1. g, h, i, j, k
2. a, b, c, d

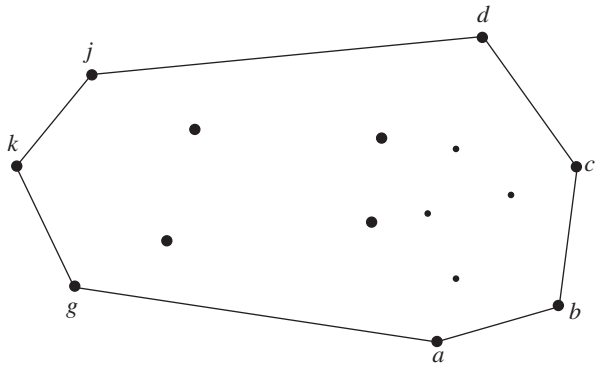
y 3. f, e.

Estas tres secuencias pueden fusionarse en una. En nuestro caso, la secuencia fusionada es

$g, h, a, b, f, c, e, d, i, j, k.$

Ahora es posible aplicar la búsqueda de Graham a esta secuencia para eliminar puntos que no pueden ser vértices del convex hull. Para el caso que se muestra en la figura 4-9, el convex hull resultante se muestra en la figura 4-11.

FIGURA 4-11 El convex hull para los puntos en la figura 4-9.



A continuación se presenta el algoritmo divide-y-vencerás para construir un convex hull.

Algoritmo 4-3 □ Un algoritmo para construir un convex hull con base en la estrategia divide-y-vencerás

Input: Un conjunto S de puntos planos.

Output: Un convex hull para S.

Paso 1. Si S no contiene más de cinco puntos, para encontrar el convex hull se usa búsqueda exhaustiva y luego se regresa.

Paso 2. Buscar una recta mediana perpendicular al eje x que divida a S en S_L y S_R; S_L está a la izquierda de S_R.

Paso 3. Recursivamente se construyen conjuntos convexos para S_L y S_R. Estos conjuntos convexos se denotan por Hull(S_L) y Hull(S_R), respectivamente.

Paso 4. Buscar un punto interior P de S_L. Buscar los vértices v_1 y v_2 de Hull (S_R) que divide los vértices de Hull (S_R) en dos secuencias de vértices que tienen ángulos polares crecientes con respecto a P. Sin pérdida de generalidad, se supone que el valor y de v_1 es mayor que el valor y de v_2. Luego se forman tres secuencias como sigue:

a) Secuencia 1: todos los vértices del convex hull de Hull(S_L) en dirección contraria al movimiento de las manecillas del reloj.

b) Secuencia 2: los vértices del convex hull de Hull(S_R) de v_2 a v_1 en dirección contraria al movimiento de las manecillas del reloj.

c) Secuencia 3: los vértices del convex hull de Hull(S_R) de v_2 a v_1 en dirección del movimiento de las manecillas del reloj.

Estas tres secuencias se fusionan y se lleva a cabo una búsqueda de Graham. Se eliminan los puntos que son reflexivos (porque producen una concavidad en un polígono convexo). Los puntos restantes forman el convex hull.

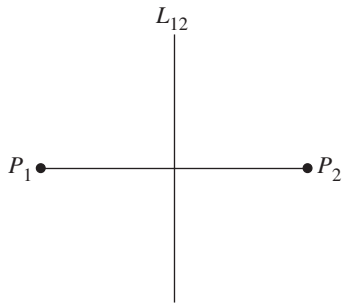
La complejidad temporal del algoritmo anterior es determinada esencialmente por el proceso de separación y el proceso de fusión. La complejidad temporal del proceso de separación es $O(n)$ porque éste es un proceso de determinación de la mediana. La complejidad temporal del proceso de fusión es $O(n)$ porque para llevar a cabo la búsqueda del punto interior, la determinación de los puntos extremos, la fusión y la búsqueda de Graham se requieren $O(n)$ pasos. Así, $T(n) = 2T(n/2) + O(n) = O(n \log n)$.

4-4 DIAGRAMAS DE VORONOI CONSTRUIDOS CON LA ESTRATEGIA DIVIDE-Y-VENCERÁS

El diagrama de Voronoi constituye, como el árbol de expansión mínima mencionado en el capítulo 3, una estructura de datos muy interesante. Esta estructura de datos puede usarse para almacenar información importante sobre los vecinos más cercanos de puntos en un plano.

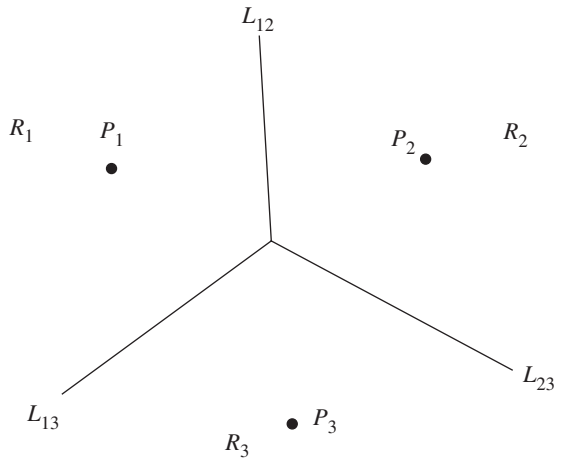
Se considerará el caso de dos puntos como se muestra en la figura 4-12. En esta figura, la recta L_{12} es una bisectriz perpendicular de la recta que une P_1 y P_2. L_{12} es el diagrama de Voronoi para P_1 y P_2. Para todo punto X situado en el semiplano que está a la izquierda (derecha) de L_{12}, el vecino más cercano de X, de entre P_1 y P_2, es $P_1(P_2)$. En cierto sentido, dado cualquier punto X, para encontrar un vecino más cercano de X no es necesario calcular la distancia entre X y P_1 y la distancia entre X y P_2. En vez de ello, sólo se requiere determinar a qué lado de L_{12} está X. Esto puede hacerse por sustitución de las coordenadas de X en L_{12}. Dependiendo del signo del resultado de esta sustitución es posible determinar dónde esta ubicado X.

FIGURA 4-12 Diagrama de Voronoi para dos puntos.



Considere la figura 4-13, donde cada L_{ij} es una bisectriz perpendicular de la recta que une P_i y P_j. Los tres hiperplanos L_{12}, L_{13} y L_{23} constituyen el diagrama de Voronoi para los puntos P_1, P_2 y P_3. Si un punto está ubicado en R_i, entonces su vecino más cercano entre P_1, P_2 y P_3 es P_i.

FIGURA 4-13 Diagrama de Voronoi para tres puntos.



Para definir el diagrama de Voronoi de un conjunto de puntos en el plano, primero se debe definir el polígono de Voronoi.

Definición

Dados dos puntos P_i y P_j en un conjunto S de n puntos, sea $H(P_i, P_j)$ el semiplano que contiene los puntos más cercanos a P_i. El polígono de Voronoi asociado con P_i es una región poligonal convexa que no tiene más de $n - 1$ lados, definida por $V(i) = \bigcap_{i \neq j} H(P_i, P_j)$.

En la figura 4-14 se muestra un polígono de Voronoi. Dado un conjunto de n puntos, el diagrama de Voronoi incluye todos los polígonos de Voronoi de estos puntos. Los vértices del diagrama de Voronoi se denominan puntos de Voronoi, y sus segmentos se denominan bordes de Voronoi. (El nombre Voronoi se refiere a un famoso matemático ruso.) En la figura 4-15 se muestra un diagrama de Voronoi para seis puntos.

FIGURA 4-14 Polígono de Voronoi.

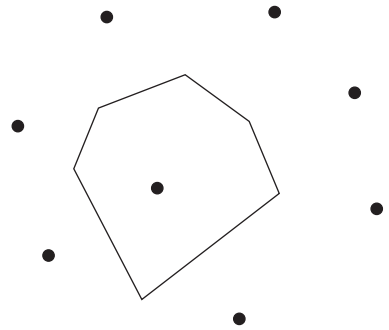
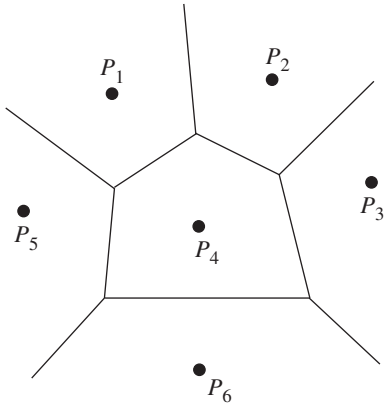


FIGURA 4-15 Diagrama de Voronoi para seis puntos.

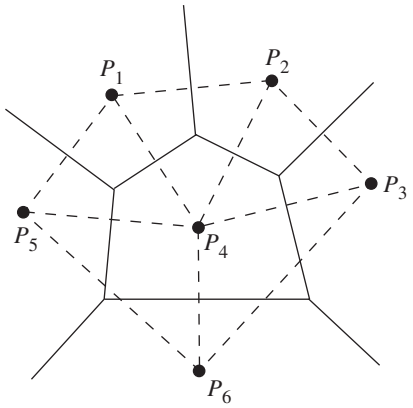


La línea recta dual de un diagrama de Voronoi se denomina triangulación Delaunay, en honor de un famoso matemático francés. En una triangulación Delaunay hay un segmento de recta que une P_i y P_j si y sólo si los polígonos de Voronoi de P_i y P_j comparten el mismo borde. Por ejemplo, para el diagrama de Voronoi en la figura 4-15, su triangulación Delaunay se muestra en la figura 4-16.

Los diagramas de Voronoi son muy útiles para varias cosas. Por ejemplo, es posible resolver el denominado problema de los pares más cercanos al extraer información del diagrama de Voronoi. A partir del diagrama de Voronoi también es posible encontrar un árbol de expansión mínima.

Un diagrama de Voronoi puede construirse a la manera divide-y-vencerás con el algoritmo de la página siguiente.

FIGURA 4-16 Una triangulación Delaunay.



Algoritmo 4-4 □ Algoritmo divide-y-vencerás para construir diagramas de Voronoi

Input: Un conjunto S de n puntos planos.
Output: El diagrama de Voronoi de S.
Paso 1. Si S contiene un solo punto, return.
Paso 2. Find una recta mediana L perpendicular al eje x que divida a S en S_L y S_R de modo que S_L (S_R) esté a la izquierda (derecha) de L y el tamaño de S_L sea igual al tamaño de S_R.
Paso 3. Recursivamente, construir diagramas de Voronoi de S_L y S_R. Denote por $VD(S_L)$ y $VD(S_R)$ estos diagramas.
Paso 4. Construir un hiperplano lineal por partes divisorio HP que sea el lugar geométrico de puntos simultáneamente más cercanos a un punto en S_L y a un punto en S_R. Eliminar todos los segmentos de $VD(S_L)$ que estén a la derecha de HP y todos los segmentos de $VD(S_R)$ que estén a la izquierda de HP. La gráfica resultante es el diagrama de Voronoi.

Este algoritmo puede entenderse al considerar la figura 4-17. Para fusionar $VD(S_L)$ y $VD(S_R)$, se observa que sólo aquellas partes de $VD(S_L)$ y $VD(S_R)$ cercanas a L interfieren entre sí. Los puntos alejados de L no son afectados por el paso de fusión, de modo que permanecen intactos.

El paso más importante al fusionar $VD(S_L)$ y $VD(S_R)$ es construir el hiperplano lineal por partes divisorio HP. Este hiperplano tiene la siguiente propiedad: si un punto P está dentro de la parte izquierda (derecha) de HP, entonces el vecino más cercano de

P debe ser un punto en $S_L(S_R)$. Véase la figura 4-17. El HP para $VD(S_L)$ y $VD(S_R)$ se muestra en la figura 4-18.

FIGURA 4-17 Dos diagramas de Voronoi después del paso 2.

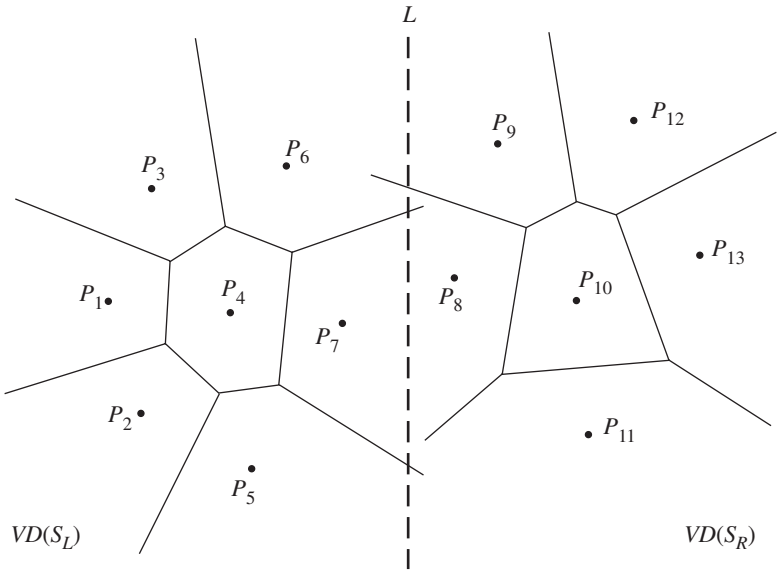
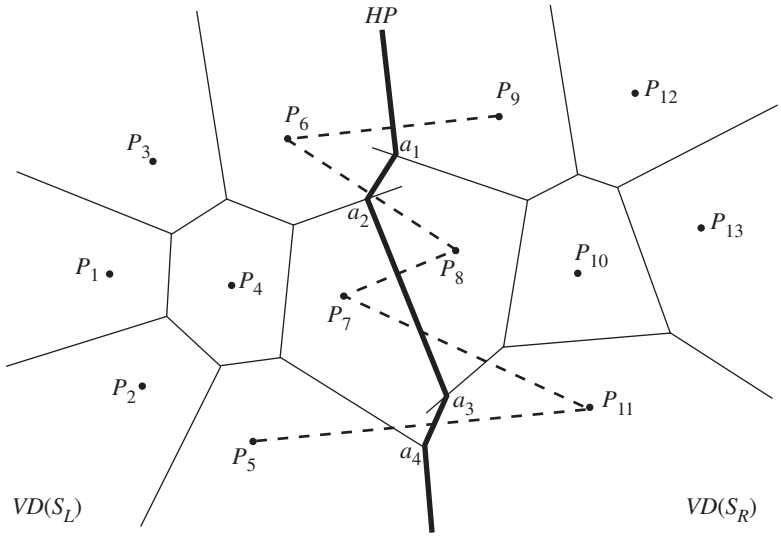
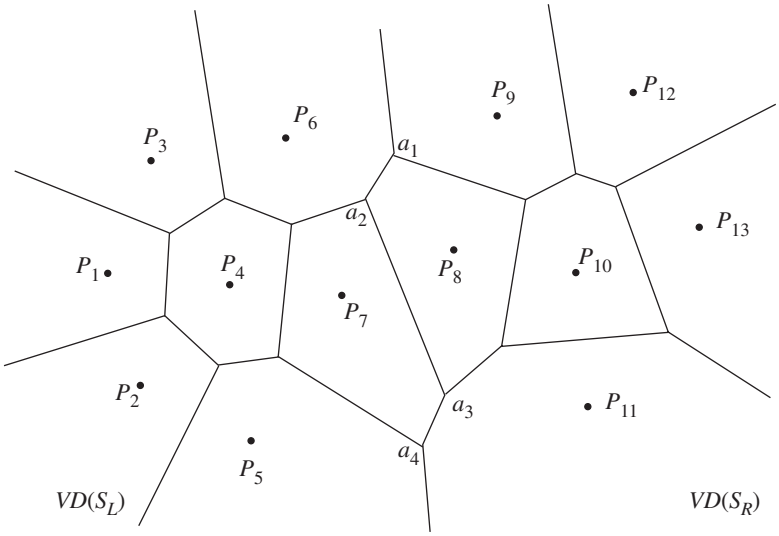


FIGURA 4-18 El hiperplano lineal por partes para el conjunto de puntos que se muestra en la figura 4-17.



Después de eliminar todos los segmentos de $VD(S_L)$ a la derecha de HP y todos los segmentos de $VD(S_R)$ a la izquierda de HP, se obtiene el diagrama de Voronoi resultante que se muestra en la figura 4-19.

FIGURA 4-19 Diagrama de Voronoi de los puntos en la figura 4-17.



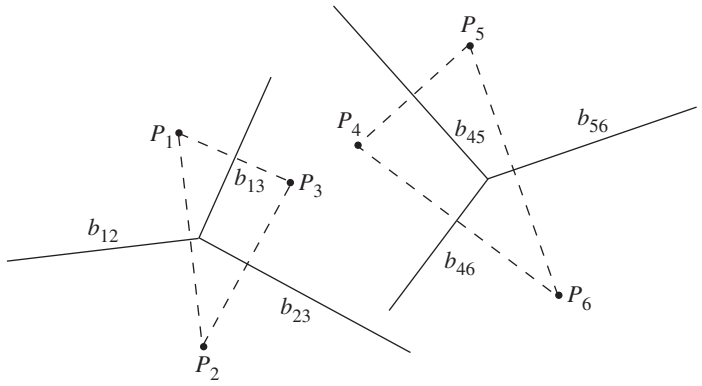
Sean Hull (S_L) y Hull (S_R) denotados los convex hull de S_L y S_R respectivamente. Existen dos segmentos de línea entre S_L y S_R que unen el Hull(S_L) y el Hull(S_R) en un convex hull. Sea $\overline{P_a P_b}$ el segmento superior, donde P_a y P_b pertenecen a S_L y S_R respectivamente. El primer paso para la construcción de HP es encontrar P_a y P_b. Se seleccionan de P_6 y P_9. Luego se traza una bisectriz perpendicular de la recta $\overline{P_6 P_9}$. Éste es el segmento inicial de HP. Luego, como se muestra en la figura 4-18, en a_1, HP se corta con la bisectriz de la recta $\overline{P_9 P_8}$. Así, se sabe que el lugar geométrico estará más cercano a P_8 que a P_9. En otras palabras, el siguiente segmento es una bisectriz de $\overline{P_6 P_8}$.

Desplazándose hacia abajo, HP corta a $VD(S_L)$ en a_2. El segmento de $VD(S_L)$ que corta a HP es la bisectriz de $\overline{P_6 P_7}$. Así, el nuevo segmento es la bisectriz de $\overline{P_7 P_8}$.

A continuación, el proceso de construcción de un diagrama de Voronoi se ilustrará con otro ejemplo. Considere los seis puntos en la figura 4-20.

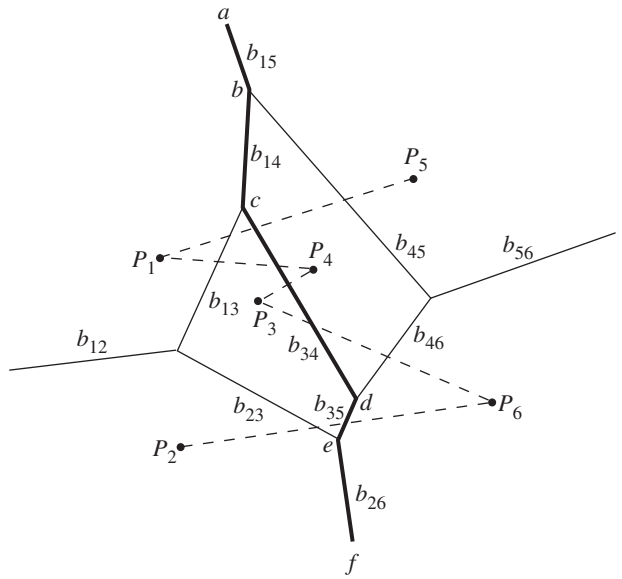
En la figura 4-20 se muestran dos diagramas de Voronoi construidos para $\{P_1, P_2, P_3\}$ y $\{P_4, P_5, P_6\}$, respectivamente, donde se han identificado todos los trazos. Por ejemplo, b_{13} es la bisectriz perpendicular de la recta $\overline{P_1 P_3}$. El proceso de construcción de un diagrama de Voronoi requiere la construcción de *conjuntos convexos*. Los dos conjuntos convexos se muestran en la figura 4-20. Ambos son triángulos. Después de que se construyen los dos conjuntos convexos, se concluye que $\overline{P_1 P_5}$ y $\overline{P_2 P_6}$ son los dos segmentos que unen los dos conjuntos convexos.

FIGURA 4-20 Otro caso que ilustra la construcción de diagramas de Voronoi.



Así, el paso de fusión empieza desde que se encuentra la bisectriz perpendicular de $\overline{P_1P_5}$, como se muestra en la figura 4-21.

FIGURA 4-21 Paso de fusión en la construcción de un diagrama de Voronoi.



Todo el paso de fusión consta de los pasos siguientes:

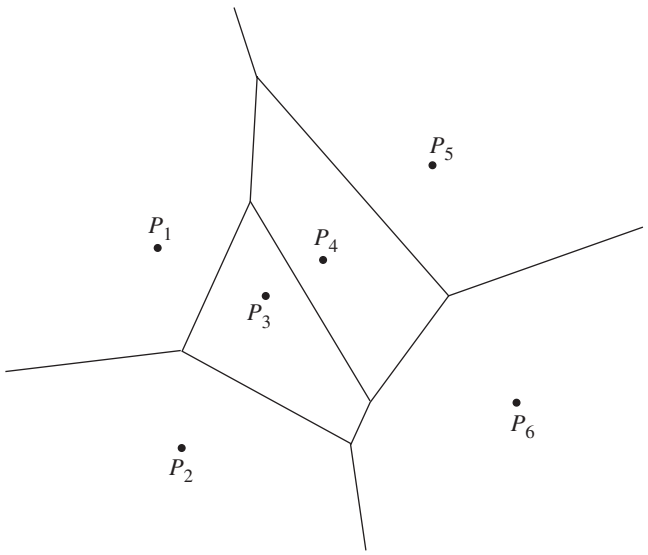
1. La bisectriz perpendicular de $\overline{P_1P_5}$ corta a b_{45} en b. Luego se encuentra la bisectriz perpendicular de $\overline{P_1P_4}$. Esta recta se identifica como b_{14}.
2. b_{14} corta a b_{13} en c. La siguiente bisectriz perpendicular de $\overline{P_3P_4}$ es b_{34}.
3. b_{34} corta a b_{46} en d. La siguiente bisectriz perpendicular de $\overline{P_3P_6}$ es b_{36}.
4. b_{36} corta a b_{23} en e. La siguiente bisectriz perpendicular de $\overline{P_2P_6}$ es b_{26}.

Debido a que P_2 y P_6 son los puntos más bajos de los dos conjuntos convexos, el trazo b_{26} se extiende al infinito y entonces se encuentra el hiperplano lineal por partes divisorio *HP*. Así, *HP* está definido por $\overline{ab}$, $\overline{bc}$, $\overline{cd}$, $\overline{de}$ y $\overline{ef}$. Si un punto cae a la derecha (izquierda) de este *HP*, entonces su vecino más cercano debe ser uno de $\{P_4, P_5, P_6\}(\{P_1, P_2, P_3\})$.

En la figura 4-21 se muestran todas las rectas cuyas bisectrices perpendiculares constituyen el hiperplano divisorio. Estas rectas se identifican como rectas discontinuas. Estos segmentos de recta son $\overline{P_5P_1}$, $\overline{P_1P_4}$, $\overline{P_4P_3}$, $\overline{P_3P_6}$ y $\overline{P_6P_2}$.

En el paso de fusión, la última parte es la eliminación de todos los $VD(S_L)$ ($VD(S_R)$) que se extiendan a la derecha (izquierda) de *HP*. El diagrama de Voronoi resultante se muestra en la figura 4-22.

FIGURA 4-22 Diagrama de Voronoi resultante.



A continuación se presenta el algoritmo para fusionar dos diagramas de Voronoi.

Algoritmo 4-5 □ Algoritmo que fusiona dos diagramas de Voronoi en un diagrama de Voronoi

Input: *a*) S_L y S_R, donde S_L y S_R están divididos por una recta perpendicular L.
b) $VD(S_L)$ y $VD(S_R)$.

Output: $VD(S)$, donde $S = S_L \cup S_R$.

Paso 1. Buscar los conjuntos convexos de S_L y S_R, que se denotan por Hull(S_L) y Hull(S_R), respectivamente. (Después se proporcionará un algoritmo para encontrar un convex hull en este caso.)

Paso 2. Buscar los segmentos $\overline{P_aP_b}$ y $\overline{P_cP_d}$ que unen a Hull(S_L) y Hull(S_R) en un convex hull (P_a y P_c pertenecen a S_L, y P_b y P_d pertenecen a S_R). Suponga que $\overline{P_aP_b}$ está por arriba de $\overline{P_cP_d}$. Hacer $x = a$, $y = b$, $SG = \overline{P_xP_y}$ y $HP = \phi$.

Paso 3. Buscar la bisectriz perpendicular de SG, que se denota por BS. Hacer $HP = HP \cup BS$. Si $SG = \overline{P_cP_d}$, ir a paso 5; en caso contrario, ir a paso 4.

Paso 4. Se deja que BS corte primero un trazo desde $VD(S_L)$ o $VD(S_R)$. Este trazo debe ser una bisectriz perpendicular de $\overline{P_xP_z}$ o $\overline{P_yP_z}$ para alguna z. Si este trazo es la bisectriz perpendicular de $\overline{P_yP_z}$, entonces hacer $SG = \overline{P_xP_z}$; en caso contrario, hacer $SG = \overline{P_zP_y}$. Ir a paso 3.

Paso 5. Se eliminan los bordes de $VD(S_L)$ que se extienden a la derecha de HP y los bordes de $VD(S_R)$ que se extienden a la izquierda de HP. La gráfica resultante es el diagrama de Voronoi de $S = S_L \cup S_R$.

A continuación se describirán en detalle las propiedades de HP, aunque antes se define la distancia entre un punto y un conjunto de puntos.

Definición

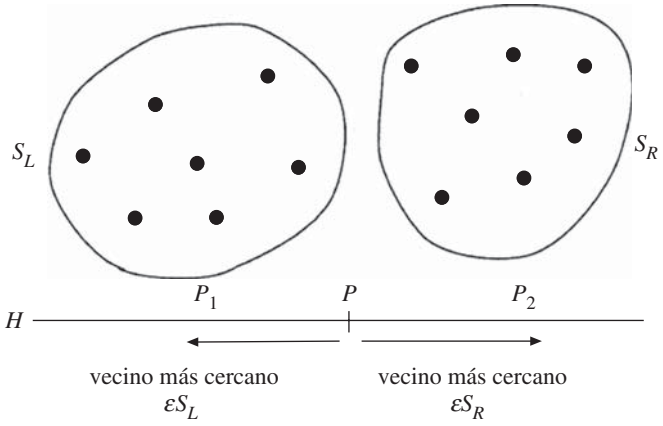
Dados un punto P y un conjunto S de puntos $P_1, P_2, \ldots, P_n$, sea P_i un vecino más cercano de P. La distancia entre P y S es la distancia entre P y P_i.

Usando esta definición puede afirmarse lo siguiente: el HP obtenido con el algoritmo 4-5 es el lugar geométrico de los puntos que preservan distancias iguales a S_L y S_R. Es decir, para cada punto P en HP, sean P_i y P_j un vecino más cercano de S_L y S_R de P, respectivamente. Entonces la distancia entre P y P_i es igual a la distancia entre P y P_j.

Por ejemplo, considere la figura 4-21. Sea P un punto sobre el segmento de recta $\overline{cd}$. Para S_L, el vecino más cercano de P es P_3 y para S_R, el vecino más cercano de P es P_4. Debido a que $\overline{cd}$ es la bisectriz perpendicular de $\overline{P_3P_4}$, cada punto en $\overline{cd}$ es equidistante a P_3 y P_4. Así, para cada punto en $\overline{cd}$, está a la misma distancia de S_L y S_R.

Al usar la propiedad anterior de HP, fácilmente se observa que *toda recta horizontal H corta a HP por lo menos una vez*. Este hecho puede verse fácilmente si se considera la figura 4-23.

FIGURA 4-23 Relación entre una recta horizontal H y S_L y S_R.

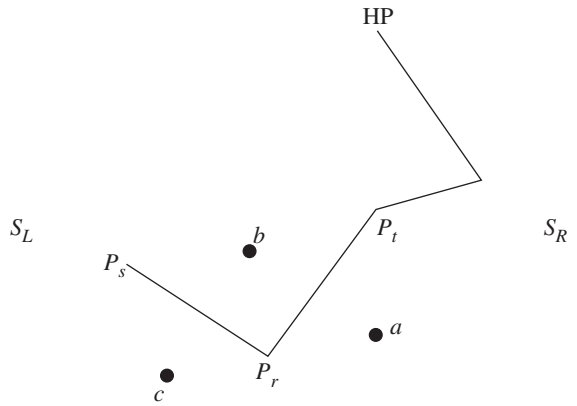


Imagine que se efectúa un desplazamiento de izquierda a derecha sobre la recta H. Al principio, para un punto como P_1, el vecino más cercano de P_1 debe ser un miembro de S_L. Imagine también que se efectúa un desplazamiento de derecha a izquierda. Para un punto como P_2, el vecino más cercano de P_2 debe ser un miembro de S_R. Debido a que H es una recta contigua, debe haber un punto P tal que a la izquierda (derecha) de P cada punto tenga como su vecino más cercano a un elemento de $S_L(S_R)$. *Así, P es equidistante a S_L y S_R. O bien, la distancia entre P y su vecino más cercano en S_L debe ser igual a la distancia entre P y su vecino más cercano en S_R.* Debido a que HP es el lugar geométrico de los puntos equidistantes a S_L y S_R, este punto P también debe estar en HP. Así, se ha demostrado que toda recta horizontal debe cortar a HP por lo menos una vez. El lector debe consultar la figura 4-21 para convencerse de la validez de la observación anterior.

Al considerar la figura 4-21 se observa que HP posee otra propiedad interesante; es monótono en y. A continuación se demostrará que *toda recta horizontal H corta a HP cuando mucho una vez.*

Suponga lo contrario. Entonces hay un punto P_r donde HP gira hacia arriba, como se muestra en la figura 4-24.

FIGURA 4-24 Ilustración de la monotonía de *HP*.



En la figura 4-24, $\overline{P_tP_r}$ es la bisectriz perpendicular de $\overline{ab}$ y $\overline{P_tP_r}$ es la bisectriz perpendicular de $\overline{bc}$. Debido a que *HP* es el lugar geométrico que separa S_L y S_R, ocurre una de dos, que *a* y *c* pertenecen a S_L y *b* pertenece a S_R o que *a* y *c* pertenecen a S_R y *b* pertenece a S_L. Ambas situaciones son imposibles porque se tiene una recta perpendicular al eje *x* para dividir a *S* en S_L y S_R. Es decir, P_r no puede existir.

Debido a que toda recta horizontal *H* corta a *HP* por lo menos una vez y cuando mucho una vez, se concluye que *toda recta horizontal H corta a HP en uno y sólo en un punto. Es decir, HP es monótono en y.*

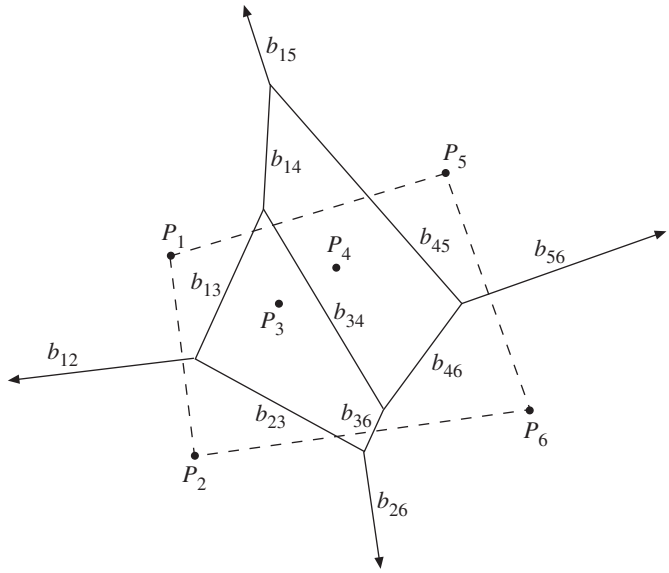
Hay otra propiedad importante de los diagramas de Voronoi que es útil para deducir la complejidad temporal del algoritmo para construir un diagrama de Voronoi. Esta propiedad es: *el número de bordes de un diagrama de Voronoi es cuando mucho* $3n - 6$*, donde n es el número de puntos*. Observe que cada diagrama de Voronoi corresponde a una triangulación Delaunay. Debido a que una triangulación Delaunay es una gráfica plana, puede contener cuando mucho $3n - 6$ bordes. Como hay una correspondencia uno a uno entre los bordes en el diagrama de Voronoi y los bordes en la triangulación Delaunay, el número de bordes en un diagrama de Voronoi es cuando mucho $3n - 6$.

Una vez que se ha obtenido la cota superior de los bordes de Voronoi, a continuación es posible deducir la cota superior de los vértices de Voronoi. Observe que cada vértice de Voronoi corresponde a un triángulo en la triangulación Delaunay. Debido a que una triangulación Delaunay es una gráfica plana, una triangulación puede considerarse como una cara de esta gráfica plana. Sean *F*, *E* y *V* el número de caras, bordes y vértices de la triangulación Delaunay. Entonces, según la relación de Euler, $F = E - V + 2$. En una triangulación Delaunay, $V = n$ y *E* es cuando mucho $3n - 6$. En consecuencia, *F* es a lo sumo $(3n - 6) - n + 2 = 2n - 4$. Es decir, *el número de vértices de Voronoi es cuando mucho* $2n - 4$.

Ya casi es posible deducir la complejidad temporal del proceso de fusión. Recuerde que en este proceso es necesario encontrar dos conjuntos convexos. El algoritmo que se presentó en el apartado 4-3 no puede aplicarse aquí porque su complejidad temporal es $O(n \log n)$, que es demasiado alta para nuestros fines. A continuación se demostrará que estos conjuntos convexos pueden encontrarse fácilmente, ya que para construir estos conjuntos convexos es posible usar $VD(S_L)$ y $VD(S_R)$.

Considere la figura 4-25 que muestra un diagrama de Voronoi con cuatro trazos infinitos; a saber, b_{12}, b_{15}, b_{56} y b_{26}. Los puntos asociados con estos trazos infinitos son P_2, P_1, P_5 y P_6. De hecho, ahora es posible construir el convex hull uniendo estos puntos, lo cual se muestra con las líneas discontinuas en la figura 4-25.

FIGURA 4-25 Construcción de un convex hull a partir de un diagrama de Voronoi.



Después de que se ha elaborado un diagrama de Voronoi, el convex hull puede construirse al analizar todos los bordes de Voronoi hasta que se encuentra un trazo infinito. Sea P_i el punto a la izquierda de este trazo infinito. Entonces P_i es el vértice de un convex hull. A continuación se analizan los bordes de Voronoi del polígono de Voronoi de P_i hasta que se encuentra un trazo infinito. El proceso anterior se repite hasta que se regresa al trazo inicial. Debido a que cada borde aparece en exactamente dos polígonos de Voronoi, ningún borde se examina más de dos veces. *En consecuencia, una vez que se construye un diagrama de Voronoi, es posible encontrar un convex hull en tiempo* $O(n)$.

A continuación se deduce la complejidad temporal del paso de fusión del algoritmo para construir un diagrama de Voronoi con base en la estrategia divide-y-vencerás:

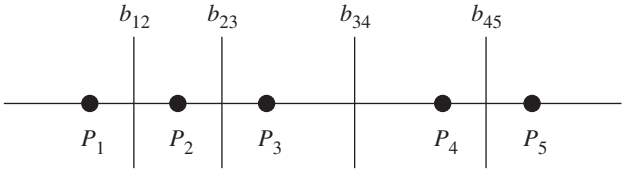
1. Los conjuntos convexos pueden encontrarse en tiempo $O(n)$ porque $VD(S_L)$ y $VD(S_R)$ ya se han construido y es posible encontrar un convex hull si se encuentran los trazos infinitos.
2. Los bordes que unen dos conjuntos convexos para formar un nuevo convex hull pueden determinarse en tiempo $O(n)$. Esto se explicó en el apartado 4-3.
3. Debido a que en $VD(S_L)$ y $VD(S_R)$ hay a lo sumo $3n - 6$ bordes y a que *HP* contiene a lo sumo n segmentos (debido a que *HP* es monótono en y), todo el proceso de construcción de *HP* requiere cuando mucho $O(n)$ pasos.

Así, el proceso de fusión del algoritmo de construcción del diagrama de Voronoi es $O(n)$. Por tanto,

$$\begin{aligned} T(n) &= 2T(n/2) + O(n) \\ &= O(n \log n). \end{aligned}$$

Se concluye que la *complejidad temporal del algoritmo divide-y-vencerás para construir un diagrama de Voronoi es O(n log n)*. A continuación se demostrará que éste es un algoritmo óptimo. Considere un conjunto de puntos en una recta. El diagrama de Voronoi de este conjunto de puntos consta de un conjunto de rectas bisectrices, como se muestra en la figura 4-26. Después que se han construido estas rectas, una exploración lineal de estos bordes de Voronoi lleva a cabo la función de ordenamiento. En otras palabras, *el problema del diagrama de Voronoi no puede ser más sencillo que el problema de ordenamiento*. En consecuencia, una cota inferior del problema del diagrama de Voronoi es $\Omega(n \log n)$ y así se concluye que el algoritmo es óptimo.

FIGURA 4-26 Diagrama de Voronoi para un conjunto de puntos en una recta.



4-5 Aplicaciones de los diagramas de Voronoi

Se ha analizado el concepto de diagramas de Voronoi y se ha demostrado cómo usar la estrategia divide-y-vencerás para construir un diagrama de Voronoi. En este apartado se demostrará que los diagramas de Voronoi poseen muchas aplicaciones interesantes. Estas aplicaciones, por supuesto, no guardan ninguna relación con la estrategia divide-y-vencerás. Mediante la presentación de aplicaciones de los diagramas de Voronoi esperamos estimular el interés del lector en la geometría computacional.

El problema euclidiano de búsqueda del vecino más cercano

La primera aplicación consiste en resolver el problema euclidiano de búsqueda del vecino más cercano, que se define como sigue: *se tienen un conjunto con n puntos planos: P_1, P_2,..., P_n y un punto de prueba P. El problema consiste en encontrar un vecino más cercano de P de entre los P_i y la distancia que se utiliza es la distancia euclidiana.*

Un método directo para resolver este problema consiste en aplicar una búsqueda exhaustiva. Este algoritmo podría ser un algoritmo $O(n)$. Al usar el diagrama de Voronoi, el tiempo de búsqueda puede reducirse a $O(\log n)$ con tiempo de procesamiento previo $O(n \log n)$.

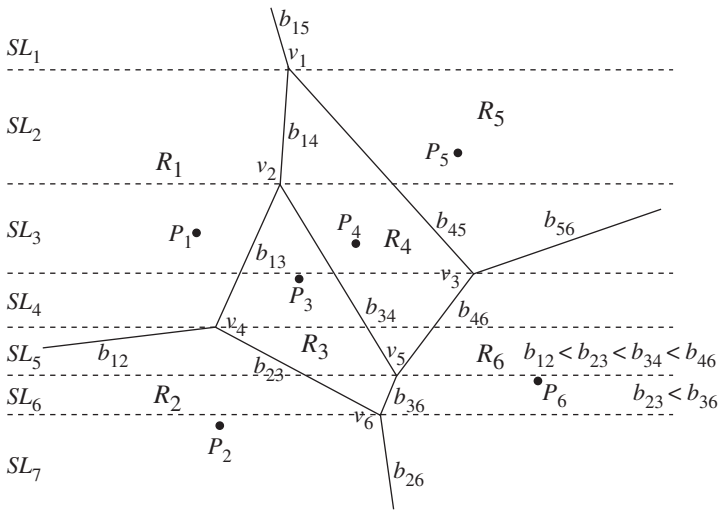
Es posible usar el diagrama de Voronoi para resolver el problema euclidiano de búsqueda del vecino más cercano debido a las propiedades fundamentales de los diagramas de Voronoi. Observe que el diagrama de Voronoi divide todo el plano en regiones $R_1, R_2,\ldots, R_n$. Dentro de cada región R_i hay un punto P_i. Si un punto de prueba cae dentro de la región R_i, entonces su vecino más cercano, de entre todos los puntos, es P_i. En consecuencia, una búsqueda exhaustiva podría evitarse transformando simplemente el problema en un problema de localización de una región. Es decir, si es posible determinar la región R_i en que se encuentra un punto de prueba, entonces se puede determinar un vecino más cercano de este punto de prueba.

Un diagrama de Voronoi es una gráfica plana. En consecuencia, es posible usar las propiedades especiales de una gráfica plana como se ilustra en la figura 4-27. En esta figura se ha vuelto a trazar el diagrama de Voronoi de la figura 4-22. Observe que hay seis vértices de Voronoi. El primer paso es ordenar estos vértices según sus valores y. Así, como se muestra en la figura 4-27, los vértices de Voronoi se identifican como $V_1, V_2,\ldots, V_6$ según sus valores decrecientes y. Para cada vértice de Voronoi se traza una recta horizontal que pase por este vértice. Tales rectas horizontales dividen todo el espacio en franjas, como se muestra en la figura 4-27.

Dentro de cada franja hay vértices de Voronoi que pueden ordenarse linealmente, que dividen nuevamente cada franja en regiones. Considere SL_5 en la figura 4-27. Dentro de SL_5 hay tres vértices de Voronoi: b_{23}, b_{34} y b_{46}. Estos vértices pueden ordenarse como

$$b_{23} < b_{34} < b_{46}.$$

FIGURA 4-27 Aplicación de los diagramas de Voronoi para resolver el problema euclidiano de búsqueda del vecino más cercano.



Estos tres bordes dividen la franja en cuatro regiones: R_2, R_3, R_4 y R_6. Si entre los bordes b_{23} y b_{34} se encuentra un punto, debe estar en R_3. Si se encuentra a la derecha de b_{46}, debe estar dentro de la región R_6. Debido a que estos bordes están ordenados, es posible usar una búsqueda binaria para determinar la ubicación de un punto de prueba en tanto concierna a los bordes de Voronoi.

Esencialmente, el algoritmo euclidiano de búsqueda del vecino más cercano consta de dos pasos fundamentales:

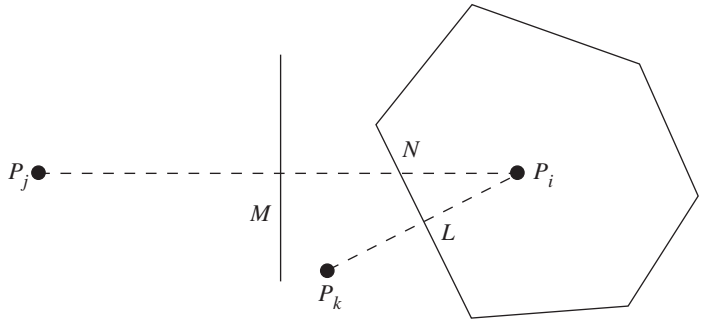
1. Efectuar una búsqueda binaria para determinar la franja en que se encuentra el punto de prueba. Debido a que cuando mucho hay $O(n)$ vértices de Voronoi, la búsqueda puede hacerse en tiempo $O(\log n)$.
2. Dentro de cada franja, realizar una búsqueda binaria para determinar la región en que se encuentra este punto. Debido a que cuando mucho hay $O(n)$ vértices de Voronoi, esta búsqueda puede hacerse en tiempo $O(\log n)$.

El tiempo total de búsqueda es $O(\log n)$. Resulta fácil darse cuenta de que el tiempo de procesamiento previo es $O(n \log n)$, que esencialmente es el tiempo necesario para construir el diagrama de Voronoi.

El problema euclidiano de todos los vecinos más cercanos

La siguiente aplicación es resolver el problema euclidiano de todos los vecinos más cercanos. Se tiene un conjunto con n puntos planos: $P_1, P_2, \ldots, P_n$. El problema euclidiano del par más cercano consiste en encontrar un vecino más cercano de todo P_i. Este problema puede resolverse fácilmente usando el diagrama de Voronoi debido a la siguiente propiedad de los diagramas de Voronoi. *Si P es un vecino más cercano de P_i, entonces P_i y P_j comparten el mismo borde de Voronoi. Además, el punto medio de $\overline{P_iP_j}$ está ubicado exactamente en este borde de Voronoi compartido comúnmente*. Demostraremos esta propiedad por contradicción. Suponga que P_i y P_j no comparten el mismo borde de Voronoi. Por la definición de polígonos de Voronoi, la bisectriz perpendicular de $\overline{P_iP_j}$ debe estar fuera del polígono de Voronoi asociado con P_i. Sea $\overline{P_iP_j}$ que corta a la bisectriz en M y a algún borde de Voronoi en N, como se ilustra en la figura 4-28.

FIGURA 4-28 Ilustración de la propiedad del vecino más cercano de los diagramas de Voronoi.



Suponga que el borde de Voronoi está entre P_i y P_k y que $\overline{P_iP_k}$ corta al borde de Voronoi en L. Entonces se tiene

$$\overline{P_iN} < \overline{P_iM}$$

y $$\overline{P_iL} < \overline{P_iN}.$$

Esto significa que

$$\overline{P_iP_k} = 2\overline{P_iL} < 2\overline{P_iN} < 2\overline{P_iM} = \overline{P_iP_j}.$$

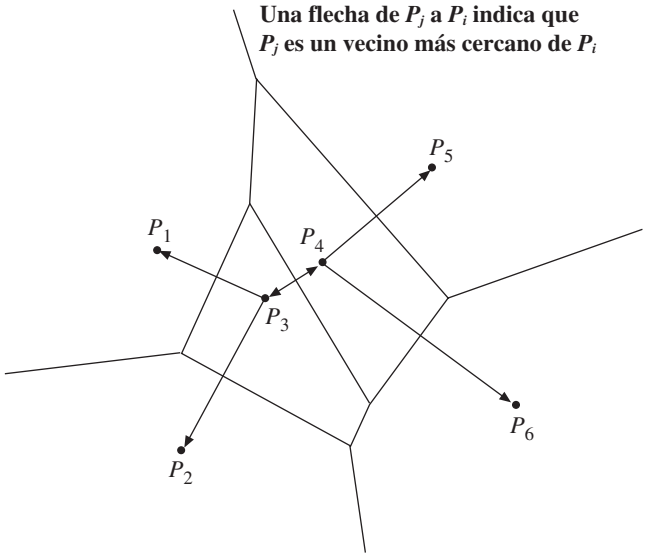
Lo anterior es imposible porque P_j es un vecino más cercano de P_i.

Dada la propiedad anterior, el problema euclidiano de todos los vecinos más cercanos puede resolverse analizando todos los bordes de Voronoi de cada polígono de

Voronoi. Debido a que cada borde de Voronoi es compartido exactamente por dos polígonos de Voronoi, ningún borde de Voronoi se analiza dos veces. Es decir, este problema euclidiano de todos los vecinos más cercanos puede resolverse en tiempo lineal una vez que se ha construido el diagrama de Voronoi. Así, este problema puede resolverse en tiempo $O(n \log n)$.

En la figura 4-29 se ha vuelto a trazar la figura 4-22. Para P_4, es necesario analizar cuatro bordes y se encontrará que el vecino más cercano de P_4 es P_3.

FIGURA 4-29 La relación de todos los vecinos más cercanos.



4-6 LA TRANSFORMADA RÁPIDA DE FOURIER

El problema de la transformada rápida de Fourier consiste en resolver lo siguiente:

$$A_j = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi ijk}{n}}, 0 \le j \le n-1$$

donde $i = \sqrt{-1}$ y $a_0, a_1, \ldots, a_{n-1}$ son números dados. Un método directo requiere $O(n^2)$ pasos. Si se aplica la estrategia divide-y-vencerás, la complejidad temporal puede reducirse a $O(n \log n)$.

Para simplificar el análisis, sea $w_n = e^{\frac{2\pi i}{n}}$. Así, la transformada de Fourier es para calcular lo siguiente:

$$A_j = a_0 + a_1 w_n^j + a_2 w_n^{2j} + \cdots + a_{n-1} w_n^{(n-1)j}.$$

Sea $n = 4$. Así, se tiene

$$\begin{aligned}
A_0 &= a_0 + a_1 + a_2 + a_3 \\
A_1 &= a_0 + a_1 w_4 + a_2 w_4^2 + a_3 w_4^3 \\
A_2 &= a_0 + a_1 w_4^2 + a_2 w_4^4 + a_3 w_4^6 \\
A_3 &= a_0 + a_1 w_4^3 + a_2 w_4^6 + a_3 w_4^9.
\end{aligned}$$

Las ecuaciones anteriores pueden volverse a agrupar como se muestra a continuación:

$$\begin{aligned}
A_0 &= (a_0 + a_2) + (a_1 + a_3) \\
A_1 &= (a_0 + a_2 w_4^2) + w_4(a_1 + a_3 w_4^2) \\
A_2 &= (a_0 + a_2 w_4^4) + w_4^2(a_1 + a_3 w_4^4) \\
A_3 &= (a_0 + a_2 w_4^6) + w_4^3(a_1 + a_3 w_4^6)
\end{aligned}$$

Debido a que $w_n^2 = w_{n/2}$ y $w_n^{n+k} = w_n^k$, se tiene

$$\begin{aligned}
A_0 &= (a_0 + a_2) + (a_1 + a_3) \\
A_1 &= (a_0 + a_2 w_2) + w_4(a_1 + a_3 w_2) \\
A_2 &= (a_0 + a_2) + w_4^2(a_1 + a_3) \\
A_3 &= (a_0 + a_2 w_4^2) + w_4^3(a_1 + a_3 w_4^2) \\
&= (a_0 + a_2 w_2) + w_4^3(a_1 + a_3 w_2).
\end{aligned}$$

Sea

$$\begin{aligned}
B_0 &= a_0 + a_2 \\
C_0 &= a_1 + a_3 \\
B_1 &= a_0 + a_2 w_2 \\
C_1 &= a_1 + a_3 w_2.
\end{aligned}$$

Entonces

$$A_0 = B_0 + w_4^0 C_0$$
$$A_1 = B_1 + w_4^1 C_1$$
$$A_2 = B_0 + w_4^2 C_0$$
$$A_3 = B_1 + w_4^3 C_1.$$

La ecuación anterior muestra que la estrategia divide-y-vencerás puede aplicarse elegantemente para resolver el problema de la transformada de Fourier. Basta calcular B_0, C_0, B_1 y C_1. Las A_j pueden calcularse fácilmente. En otras palabras, una vez que se obtiene A_0, A_2 puede obtenerse de inmediato. De manera semejante, una vez que se obtiene A_1, A_3 puede obtenerse directamente.

Pero, ¿qué ocurre con las B_i y con las C_i? Observe que las B_i son la transformada de Fourier de los datos de entrada con número impar, y que las C_i representan la transformada de Fourier de los datos de entrada con número par. Esto constituye la base de la aplicación de la estrategia divide-y-vencerás al problema de la transformada de Fourier. Un problema grande se divide en dos subproblemas, que se resuelven de manera recursiva y luego se fusionan las soluciones.

Considere A_j en el caso general.

$$\begin{aligned} A_j &= a_0 + a_1 w_n^j + a_2 w_n^{2j} + \cdots + a_{n-1} w_n^{(n-1)j} \\ &= (a_0 + a_2 w_n^{2j} + a_4 w_n^{4j} + \cdots + a_{n-2} w_n^{(n-2)j}) \\ &\quad + w_n^j (a_1 + a_3 w_n^{2j} + a_5 w_n^{4j} + \cdots + a_{n-1} w_n^{(n-2)j}) \\ &= (a_0 + a_2 w_{n/2}^j + a_4 w_{n/2}^{2j} + \cdots + a_{n-2} w_{n/2}^{\frac{(n-2)j}{2}}) \\ &\quad + w_n^j (a_1 + a_3 w_{n/2}^j + a_5 w_{n/2}^{2j} + \cdots + a_{n-1} w_{n/2}^{\frac{(n-2)j}{2}}). \end{aligned}$$

B_j y C_j se definen como sigue:

$$B_j = a_0 + a_2 w_{n/2}^j + a_4 w_{n/2}^{2j} + \cdots + a_{n-2} w_{n/2}^{\frac{(n-2)j}{2}}$$

y $$C_j = a_1 + a_3 w_{n/2}^j + a_5 w_{n/2}^{2j} + \cdots + a_{n-1} w_{n/2}^{\frac{(n-2)j}{2}}.$$

Entonces

$$A_j = B_j + w_n^j C_j.$$

También puede demostrarse que

$$A_{j+n/2} = B_j + w_n^{j+n/2} C_j.$$

Para $n = 2$, la transformada de Fourier es como sigue:

$$A_0 = a_0 + a_1$$
$$A_1 = a_0 - a_1.$$

A continuación se presenta el algoritmo de la transformada rápida de Fourier basado en el método divide-y-vencerás:

Algoritmo 4-6 □ Un algoritmo de la transformada rápida de Fourier basado en la estrategia divide-y-vencerás

Input: $a_0, a_1, \ldots, a_{n-1}, n = 2^k$.

Output: $A_j = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi ijk}{n}}$ para $j = 0, 1, 2, \ldots, n-1$.

Paso 1. Si $n = 2$,

$A_0 = a_0 + a_1$

$A_1 = a_0 - a_1$

Return.

Paso 2. Recursivamente buscar los coeficientes de la transformada de Fourier de $a_0, a_2, \ldots, a_{n-2}(a_1, \ldots, a_{n-1})$. Los coeficientes se denotan por $B_0, B_1, \ldots, B_{n/2}(C_0, C_1, \ldots, C_{n/2})$.

Paso 3. Para $j = 0$ hasta $j = \frac{n}{2} - 1$

$A_j = B_j + w_n^j C_j$

$A_{j+n/2} = B_j + w_n^{j+n/2} C_j.$

Resulta evidente que la complejidad temporal del algoritmo anterior es $O(n \log n)$. Hay una transformada inversa de Fourier que transforma $A_0, A_1, \ldots, A_{n-1}$ de vuelta en $a_0, a_1, \ldots, a_{n-1}$ como sigue:

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} A_k e^{\frac{-2\pi ijk}{n}} \quad \text{para} \quad j = 0, 1, 2, \ldots, n-1.$$

Se considerará un ejemplo. Sean $a_0 = 1$, $a_1 = 0$, $a_2 = -1$, $a_3 = 0$. Entonces

$$B_0 = a_0 + a_2 = 1 + (-1) = 0$$
$$B_1 = a_0 - a_2 = 1 - (-1) = 2$$
$$C_0 = a_1 + a_3 = 0 + 0 = 0$$
$$C_1 = a_1 - a_3 = 0 - 0 = 0$$
$$w_4 = e^{\frac{2\pi i}{4}} = i$$
$$w_4^2 = -1$$
$$w_4^3 = -i.$$

Así,

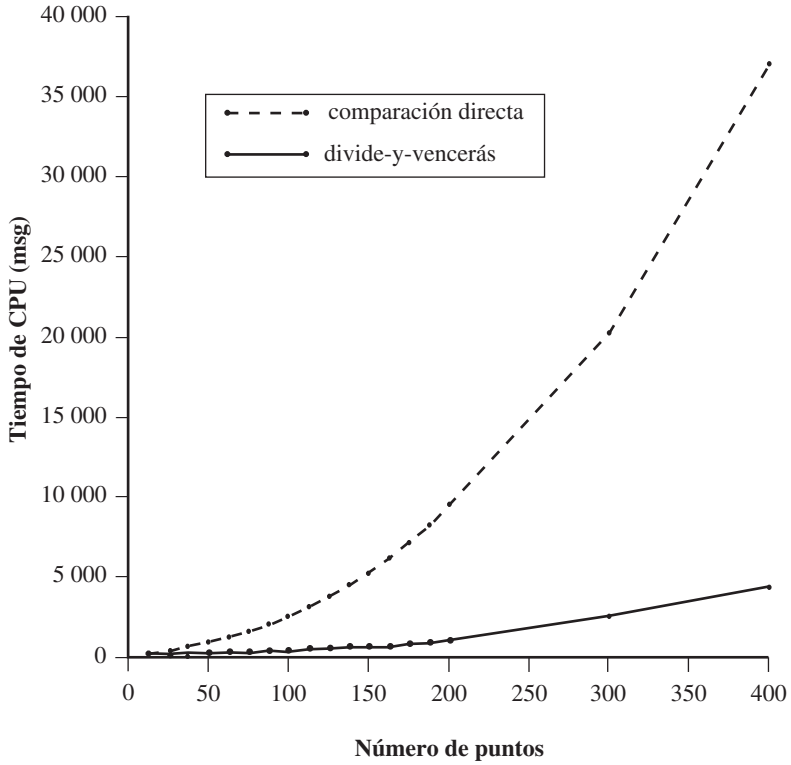
$$A_0 = B_0 + w_4^0 C_0 = B_0 + C_0 = 0 + 0 = 0$$
$$A_1 = B_1 + w_4 C_1 = 2 + 0 = 2$$
$$A_2 = B_0 + w_4^2 C_0 = 0 - 0 = 0$$
$$A_3 = B_1 + w_4^3 C_1 = 2 + 0 = 2.$$

Debe ser fácil ver que la transformada inversa de Fourier debe transformar correctamente las A_i de vuelta en las a_i.

4-7 LOS RESULTADOS EXPERIMENTALES

Para demostrar que la estrategia divide-y-vencerás es útil, se implementó el algoritmo del par más cercano basado en los algoritmos divide-y-vencerás, y el algoritmo de comparación directa que examina exhaustivamente cada par de puntos. Ambos programas se implementaron en una computadora personal IBM. Los resultados experimentales se resumen en la figura 4-30. Como se muestra en esa figura, mientras n es pequeña el método directo se desempeña mejor. No obstante, a medida que n se hace grande, el algoritmo divide-y-vencerás es mucho más rápido. Por ejemplo, cuando n es igual a 7 000, el algoritmo divide-y-vencerás es casi 200 veces más rápido que el método directo. Los resultados son bastante predecibles porque la complejidad temporal del método directo es $O(n^2)$ y la complejidad temporal del algoritmo divide-y-vencerás es $O(n \log n)$.

FIGURA 4-30 Resultados experimentales del problema de encontrar el par más cercano.



4-8 NOTAS Y REFERENCIAS

La estrategia divide-y-vencerás suele aplicarse para resolver problemas de geometría computacional. Consulte a Preparata y Shamos (1985) y a Lee y Preparata (1984). La estrategia divide-y-vencerás también se analiza en Horowitz y Sahni (1978) y en Aho, Hopcroft y Ullman (1974).

Para un análisis completo del problema de máximos, consulte a Pohl (1972). El problema del par más cercano y el problema de cómo encontrar los máximos fueron resueltos por Bentley (1980).

Los diagramas de Voronoi fueron estudiados primero por Voronoi (1908). De nuevo, más información sobre los diagramas de Voronoi puede consultarse en Preparata y Shamos (1985). Para una generalización de los diagramas de Voronoi a órdenes y dimensiones superiores, consulte las obras de Lee (1982) y Brown (1979).

En Levcopoulos y Lingas (1987) y Preparata y Shamos (1985) pueden encontrarse diferentes algoritmos de conjuntos convexos. El algoritmo de Graham fue propuesto por Graham (1972). Aplicando la estrategia divide-y-vencerás también puede encontrarse un convex hull tridimensional. Esto fue indicado por Preparata y Hong (1977).

El hecho de que la transformada de Fourier puede resolverse con el método de divide-y-vencerás fue señalado por Cooley y Tukey (1965). Gentleman y Sande (1966) analizan aplicaciones de la transformada rápida de Fourier. El libro de Brigham (Brigham, 1974) está dedicado totalmente a la transformada rápida de Fourier.

La estrategia divide-y-vencerás puede aplicarse para obtener un algoritmo eficiente de multiplicación matricial. Consulte la obra de Strassen (1969).

4-9 BIBLIOGRAFÍA ADICIONAL

La estrategia divide-y-vencerás sigue siendo un tema interesante para muchos investigadores. Es especialmente eficaz para geometría computacional. Para investigación adicional, se recomiendan los siguientes artículos: Aleksandrov y Djidjev (1996); Bentley (1980); Bentley y Shamos (1978); Blankenagel y Gueting (1990); Bossi, Cocco y Colussi (1983); Chazelle, Drysdale y Lee (1986); Dwyer (1987); Edelsbrunner, Maurer, Preparata, Rosenberg, Welzl y Wood (1982); Gilbert, Hutchinson y Tarjan (1984); Gueting (1984); Gueting y Schilling (1987); Karp (1994); Kumar, Kiran y Pandu (1987); Lopez y Zapata (1994); Monier (1980); Oishi y Sugihara (1995); Reingold y Supowit (1983); Sykora y Vrto (1993); Veroy (1988); Walsh (1984), y Yen y Kuo (1997).

Para algunos artículos bastante interesantes y de reciente publicación, consulte Abel (1990); Derfel y Vogl (2000); Dietterich (2000); Even, Naor, Rao y Schieber (2000); Fu (2001); Kamidoi, Wakabayashi y Yoshida (2002); Lee y Sheu (1992); Liu (2002); Lo, Rajopadhye, Telle y Zhong (1996); Melnik y Garcia-Molina (2002); Messinger, Rowe y Henry (1991); Neogi y Saha (1995); Rosler (2001); Rosler y Ruschendorf (2001); Roura (2001); Tisseur y Dongarra (1999); Tsai y Katsaggelos (1999); Verma (1997); Wang (1997); Wang (2000); Wang, He, Tang y Wee (2003); Yagle (1998), y Yoo, Smith y Gopalarkishnan (1997).

Ejercicios

4.1 La búsqueda binaria, ¿usa la estrategia divide-y-vencerás?

4.2 Para multiplicar directamente dos números de n bits u y v se requieren $O(n^2)$ pasos. Al aplicar el método divide-y-vencerás, el número puede separarse en dos partes iguales y calcular el producto aplicando el método siguiente:

$$uv = (a \cdot 2^{n/2} + b) \cdot (c \cdot 2^{n/2} + d) = ac \cdot 2^n + (ad + bc) \cdot 2^{n/2} + bd.$$

Si $ad + bc$ se calcula como $(a + b)(c + d) - ac - bd$, ¿cuál es el tiempo de cálculo?

4.3 Demuestre que en quick sort, la pila máxima necesaria es $O(\log n)$.

4.4 Implemente el algoritmo de la transformada rápida de Fourier con base en el método divide-y-vencerás. Compárelo con el método directo.

4.5 Implemente el algoritmo para encontrar rangos con base en el método divide-y-vencerás. Compárelo con el método directo.

4.6 Sea $T\left(\frac{n^2}{2^r}\right) = nT(n) + bn^2$, donde r es un entero y $r \leq 1$. Encuentre $T(n)$.

4.7 Lea la sección 3-7 del libro de Horowitz y Sahni (1978) para conocer el método de multiplicación matricial de Strassen con base en el algoritmo divide-y-vencerás.

4.8 Sea

$$T(n) = \begin{cases} b & \text{para } n = 1 \\ aT(n/c) + bn & \text{para } n > 1 \end{cases}$$

donde a, b y c son constantes no negativas.

Demuestre que si n es una potencia de c, entonces

$$T(n) = \begin{cases} O(n) & \text{si } a < c \\ O(n \log_a n) & \text{si } a = c \\ O(n^{\log_c a}) & \text{si } a > c. \end{cases}$$

4.9 Demuestre que si $T(n) = mT(n/2) + an^2$, entonces $T(n)$ es satisfecho por

$$T(n) = \begin{cases} O(n^{\log m}) & \text{si } m > 4 \\ O(n^2 \log n) & \text{si } m = 4 \\ O(n^2) & \text{si } m < 4. \end{cases}$$

4.10 Una clase muy especial de algoritmo de ordenamiento, basado también en el algoritmo divide-y-vencerás, es el ordenamiento de fusión impar-par (odd-even merge sorting), inventado por Batcher (1968). Lea la sección 7.4 del libro de Liu (1977) para conocer este algoritmo de ordenamiento. Este algoritmo, ¿es idóneo como algoritmo secuencial? ¿Por qué? (Éste es un famoso algoritmo de ordenamiento paralelo.)

4.11 Diseñe un algoritmo con tiempo $O(n \log n)$ para encontrar la subsecuencia monótona creciente más larga de una secuencia de n números.